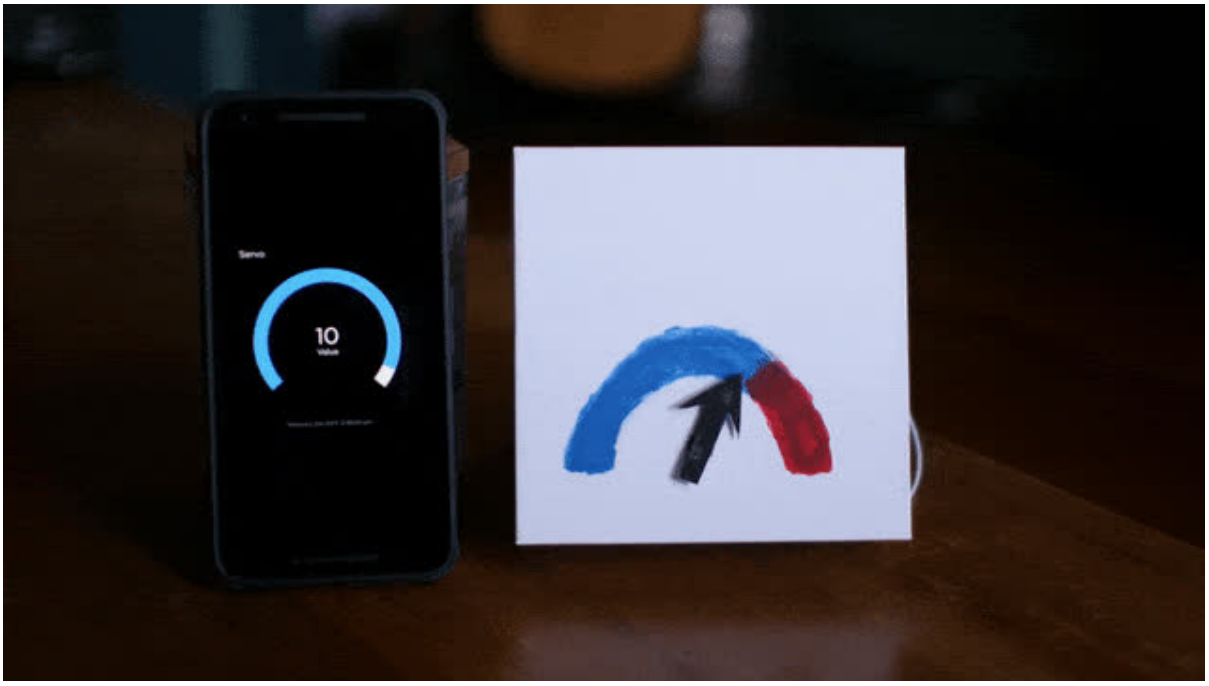
Adafruit IO Basics: Servo

Created by Todd Treece

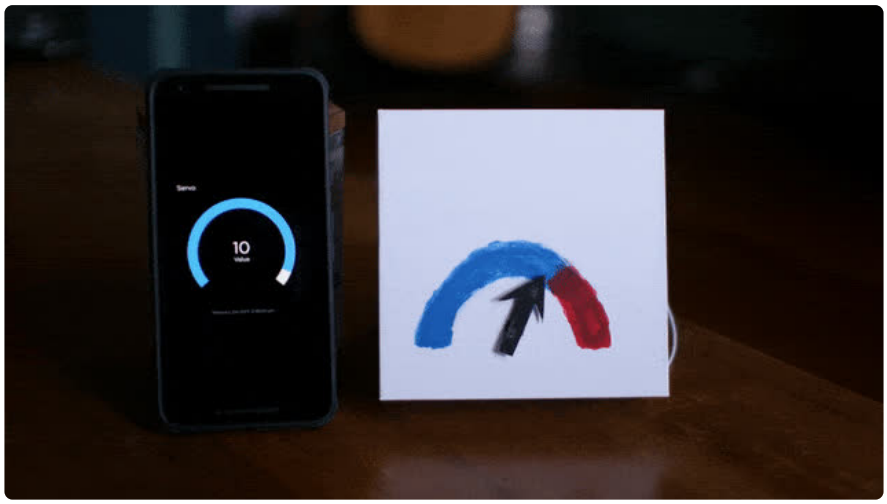


https://learn.adafruit.com/adafruit-io-basics-servo

Last updated on 2024-06-03 02:05:01 PM EDT

Table of Contents

Overview



This guide is part of a series of guides that cover the basics of using Adafruit IO. It will show you how to wirelessly control a servo from Adafruit IO.

If you haven't worked your way through the Adafruit IO feed and dashboard basics guides, you should do that before continuing with this guide so you have a basic understanding of Adafruit IO.

- Adafruit IO Basics: Feeds
- Adafruit IO Basics: Dashboards

You should go through the setup guides associated with your selected set of hardware, and make sure you have internet connectivity with the device before continuing. The following links will take you to the guides for your selected platform.

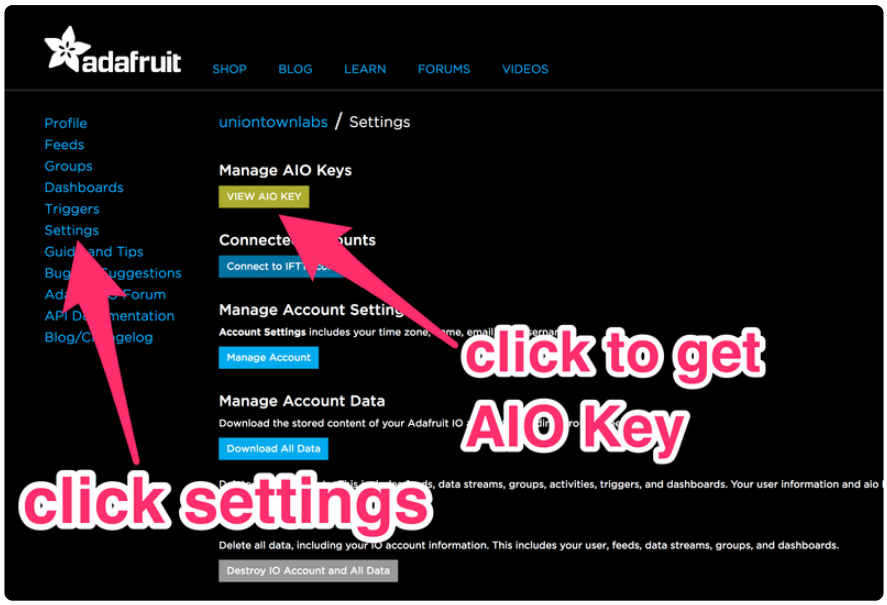
- Adafruit Feather HUZZAH ESP8266 Setup Guide

If you have went through all of the prerequisites for your selected hardware, you are now ready to move on to the Adafruit IO setup steps that are common between all of the hardware choices for this project. Let's get started!

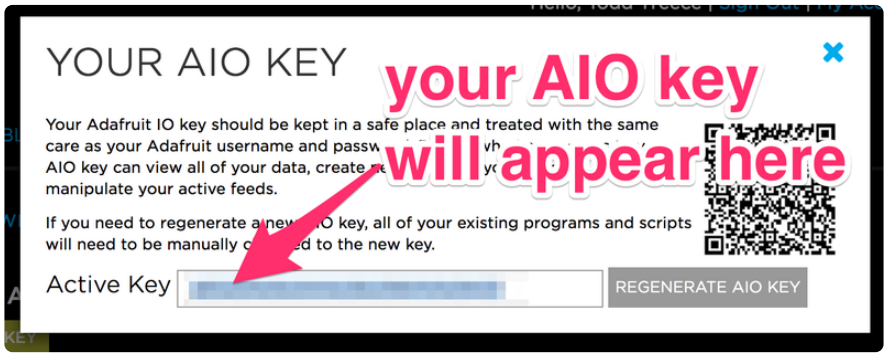
Adafruit IO Setup

The first thing you will need to do is to login to Adafruit IO and visit the **Settings** page.

Click the **VIEW AIO KEY** button to retrieve your key.



A window will pop up with your Adafruit IO. Keep a copy of this in a safe place. We'll need it later.



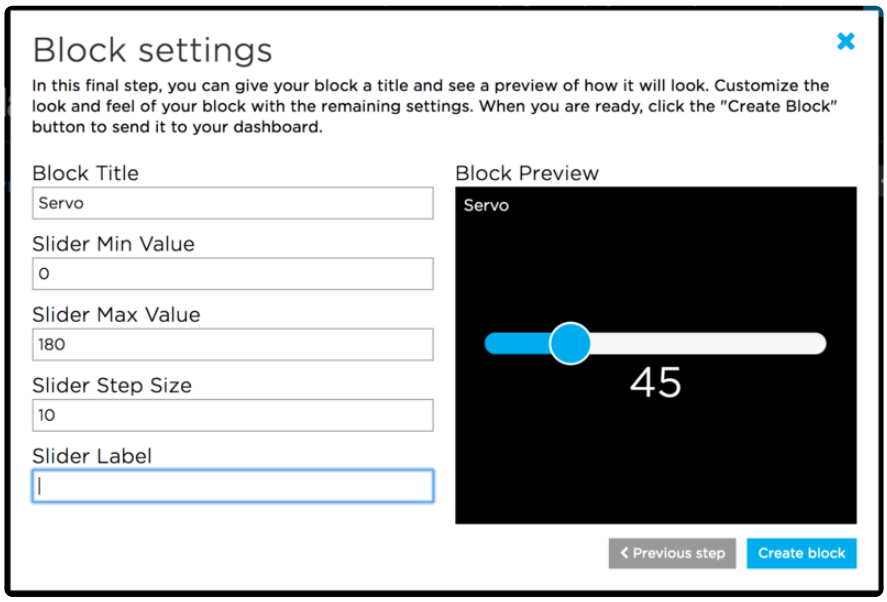
Creating the Servo Feed

Next, you will need to create a feed called **Servo**. If you need help getting started with creating feeds on Adafruit IO, check out the Adafruit IO Feed Basics guide (https://adafru.it/ioA).

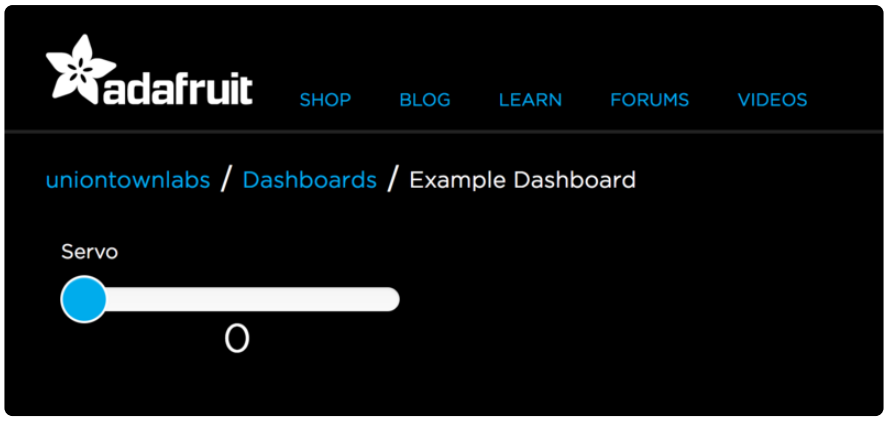
Adding the Slider Block

Next, add a new Slider Block to a new or existing dashboard. Name the block whatever you would like, and set **min value to 0** and **max value to 180**. Make sure you have selected the **Servo** feed as the data source for the slider.

If you need help getting started with Dashboards on Adafruit IO, check out the Adafruit IO Dashboard Basics guide (https://adafru.it/f5m).



When you are finished editing the form, click Create Block to add the new block to the dashboard.



Next, we will look at wiring the circuit.

Wiring

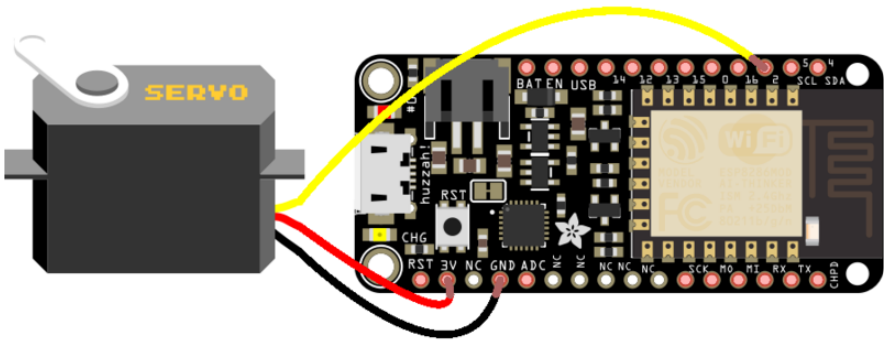
Arduino Wiring

You will need the following parts for this tutorial:

- **1x** Adafruit IO compatible Feather
- **3x** jumper wires
- **1x** micro servo

You will need to connect the following pins to the servo using the three jumper wires:

- Feather **GND** to the **brown** or **black** servo wire
- Feather **3V** to the **red** servo wire
- Feather **Pin 2** to the **yellow** servo wire

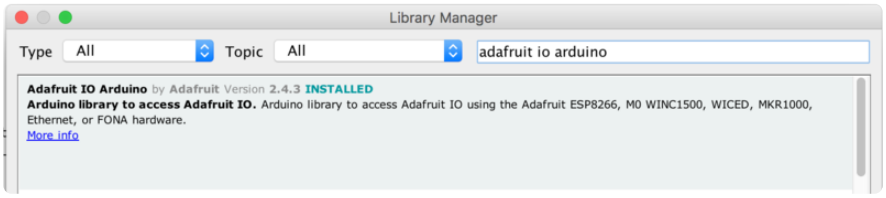


Arduino Setup

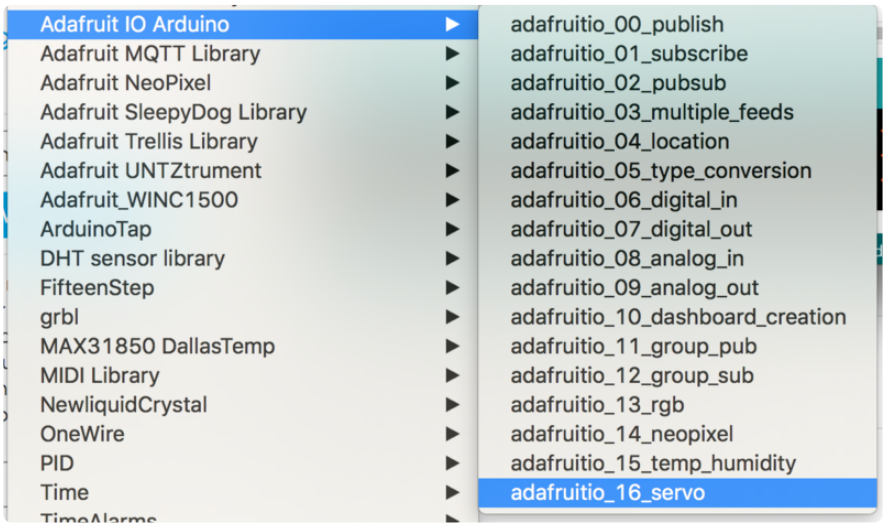
You should go through the setup guides associated with your selected set of hardware, and make sure you have internet connectivity with the device before continuing. The following links will take you to the guides for your selected platform.

- Adafruit Feather HUZZAH ESP8266 Setup Guide

You will need to make sure you have at least **version 2.4.3** of the Adafruit IO Arduino library installed before continuing.



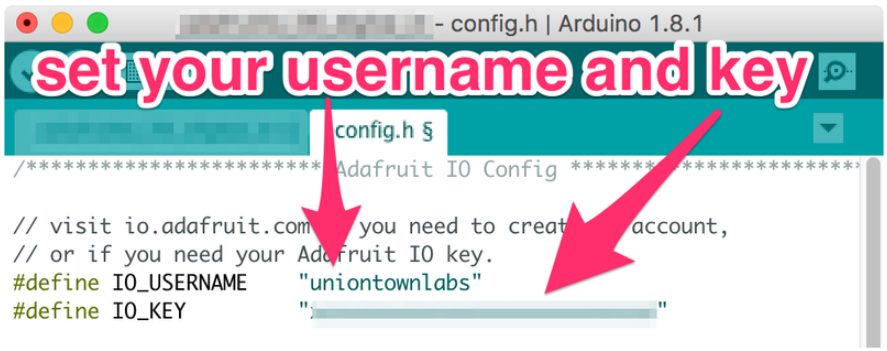
For this example, you will need to open the **adafruitio_16_servo** example in the **Adafruit IO Arduino** library.



Next, we will look at the network configuration options in the sketch.

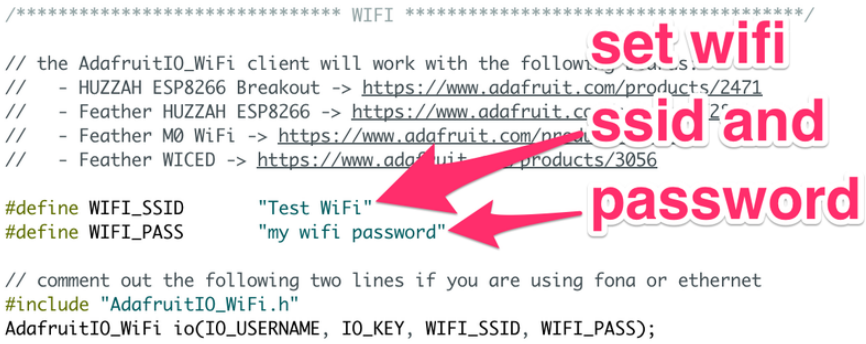
Arduino Network Config

To configure the network settings, click on the **config.h** tab in the sketch. You will need to set your Adafruit IO username in the **IO_USERNAME** define, and your Adafruit IO key in the **IO_KEY** define.



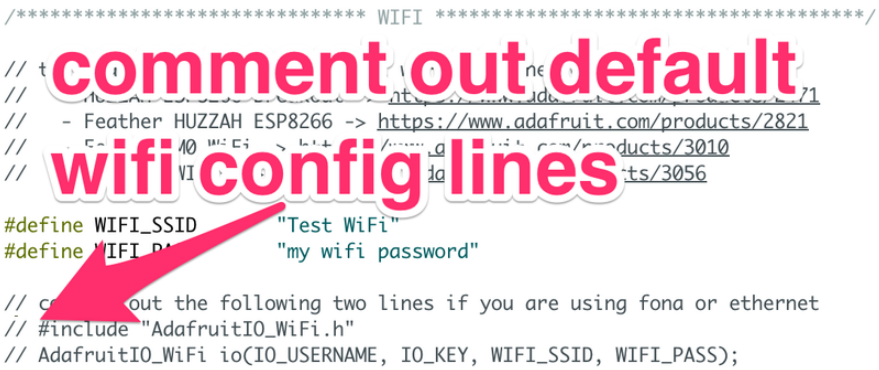
WiFi Config

WiFi is enabled by default in **config.h** so if you are using one of the supported WiFi boards, you will only need to modify the **WIFI_SSID** and **WIFI_PASS** options in the **config.h** tab.



FONA Config

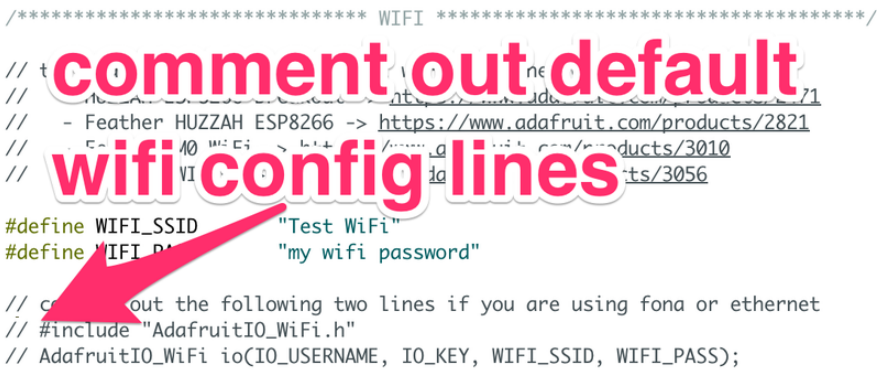
If you wish to use the FONA 32u4 Feather to connect to Adafruit IO, you will need to first comment out the WiFi support in **config.h**



Next, remove the comments from both of the FONA config lines in the FONA section of **config.h** to enable FONA support.

Ethernet Config

If you wish to use the Ethernet Wing to connect to Adafruit IO, you will need to first comment out the WiFi support in **config.h**



Next, remove the comments from both of the Ethernet config lines in the Ethernet section of **config.h** to enable Ethernet Wing support.

Next, we will look at how the example sketch works.

Arduino Code

The **adafruitio_16_servo** example uses **pin 2** by default, and that can be modified by changing the **SERVO_PIN** define at the top of the sketch.

```
// pin used to control the servo
#define SERVO_PIN 2
```

The next chunk of code sets up an instance of the Servo class, and also an instance of the Adafruit IO Feed class for a feed called **servo**.

```
// create an instance of the servo class
Servo servo;

// set up the 'servo' feed
AdafruitIO_Feed *servo_feed = io.feed("servo");
```

In the setup function, we attach a function called **handleMessage** to the **servo_feed**, which will be called whenever your device receives messages for that feed. We also tell the servo class which pin we are using with the `servo.attach()` method.

The code will wait until you have a valid connection to Adafruit IO before continuing with the sketch. If you have any issues connecting, check **config.h** for any typos in your username or key.

```
void setup() {

  // start the serial connection
  Serial.begin(115200);

  // wait for serial monitor to open
  while(! Serial);

  // tell the servo class which pin we are using
  servo.attach(SERVO_PIN);

  // connect to io.adafruit.com
  Serial.print("Connecting to Adafruit IO");
  io.connect();

  // set up a message handler for the 'servo' feed.
  // the handleMessage function (defined below)
  // will be called whenever a message is
  // received from adafruit io.
  servo_feed-&gt;onMessage(handleMessage);

  // wait for a connection
  while(io.status() &lt; AIO_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  // we are connected
  Serial.println();
  Serial.println(io.statusText());

}
```

Next, we have the main `loop()` function. The first line of the loop function calls `io.run();` this line will need to be present at the top of your loop in every sketch. It helps keep your device connected to Adafruit IO, and processes any incoming data.

```
void loop() {
  // io.run(); is required for all sketches.
  // it should always be present at the top of your loop
```

```
  // function. it keeps the client connected to
  // io.adafruit.com, and processes any incoming data.
  io.run();
}
```

The final chunk of code is the **handleMessage** function. This is the function that is called whenever **servo_feed** gets a message.

We use the `data->toInt()` function to convert the incoming data to an **int**, and set the angle of the servo to that value using `servo->write()`. We also check to make sure that the incoming angle value is not less than 0, or greater than 180.

```
      // this function is called whenever a 'servo' message
// is received from Adafruit IO. it was attached to
// the servo feed in the setup() function above.
void handleMessage(AdafruitIO_Data *data) {

  // convert the data to integer
  int angle = data-&gt;toInt();

  // make sure we don't exceed the limit
  // of the servo. the range is from 0
  // to 180.
  if(angle &lt; 0)
    angle = 0;
  else if(angle &gt; 180)
    angle = 180;

  servo.write(angle);

}
```

If you would like your servo to switch directions, you can subtract the angle from 180, and write the result to the servo.

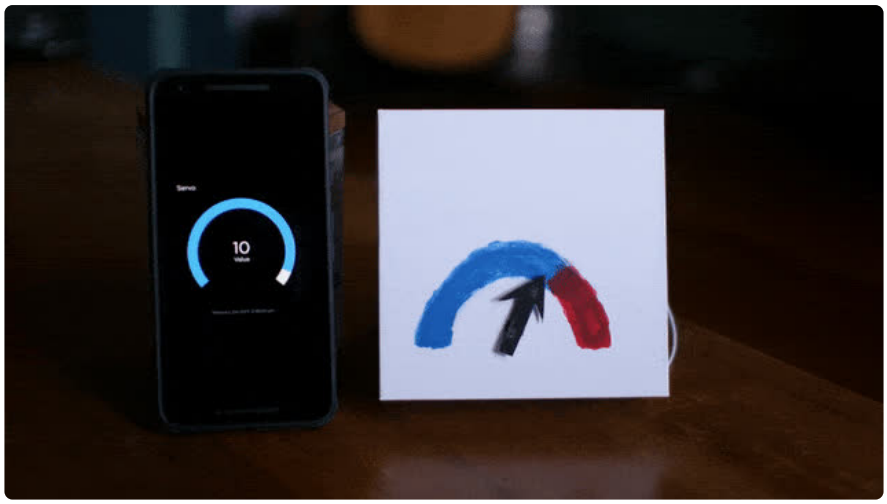
```
 servo.write(180 - angle);
```

Upload the sketch to your board, and open the Arduino Serial Monitor. Your board should now connect to Adafruit IO.

```
Connecting to Adafruit IO....

Adafruit IO connected.
```

Change the slider value on your Adafruit IO dashboard, and you should see the servo change position depending on the value you send.




```
"""
`servo.py`
================================================================================

Control a servo with Adafruit IO
Tutorial Link: https://learn.adafruit.com/adafruit-io-basics-servo

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Author(s): Brent Rubell for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.
All text above must be included in any redistribution.

Dependencies:
    - Adafruit_Blinka
        (https://github.com/adafruit/Adafruit_Blinka)
    - Adafruit_CircuitPython_PCA9685
        (https://github.com/adafruit/Adafruit_CircuitPython_PCA9685)
    - Adafruit_CircuitPython_Motor
        (https://github.com/adafruit/Adafruit_CircuitPython_Motor)
"""

# import system libraries
import time

# import Adafruit Blinka
from board import SCL, SDA
from busio import I2C

# import the PCA9685 module.
from adafruit_pca9685 import PCA9685

# import the adafruit_motor library
from adafruit_motor import servo

# import Adafruit IO REST client
from Adafruit_IO import Client, Feed, RequestError

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_AIO_USERNAME'

# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_AIO_KEY'
```

```
# Create an instance of the REST client.
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

try: # if we have a 'servo' feed
    servo_feed = aio.feeds('servo')
except RequestError: # create a servo feed
    feed = Feed(name='servo')
    servo_feed = aio.create_feed(feed)

# Create the I2C bus interface.
i2c_bus = I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c_bus)

# Set the PWM frequency to 50hz.
pca.frequency = 50
SERVO_CHANNEL = 0

# counter variable for the last servo angle
prev_angle = 0

# set up the servo on PCA channel 0
my_servo = servo.Servo(pca.channels[SERVO_CHANNEL])


while True:
    # grab the `servo` feed value
    servo_angle = aio.receive(servo_feed.key)
    if servo_angle.value != prev_angle:
        print('received <- ', servo_angle.value, 'Degrees')
        # write the servo to the feed-specified angle
        my_servo.angle = int(servo_angle.value)
    prev_angle = servo_angle.value
    # timeout so we don't flood IO with requests
    time.sleep(0.5)
```

Python Wiring

We're going to use a combination of the Adafruit IO Client Library and Adafruit's CircuitPython to control a Raspberry Pi over Adafruit IO.

1 x Raspberry Pi 3 - Model B+ https://www.adafruit.com/product/3775

The Raspberry Pi is a small linux board compatible with Adafruit IO projects.

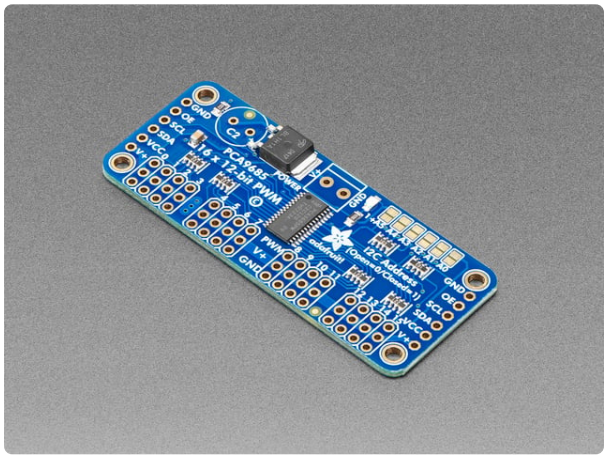
If you're following along with a Raspberry Pi (https://adafru.it/ejq), we're going to use a T-Cobbler Plus for the IO Basics Projects. This add-on prototyping board lets you easily connect a Raspberry Pi (Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3) to a solderless breadboard:

Assembled Pi T-Cobbler Plus - GPIO Breakout
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...
https://www.adafruit.com/product/2028

Want to create an automatic fish-feeder, a door-lock system with vibration-feedback, or maybe you want to just chain a bunch of lights and motors together and control them with Adafruit IO?

You'll need a few PWM outputs. This guide requires only one for the servo.

The Raspberry Pi is limited to one PWM output. While we could use this PWM output for the servo, we're going to use the Adafruit 16-Channel 12-bit PWM/Servo driver (http://adafru.it/815). This board can be used to control up to 16 PWM outputs. This means you can have a bunch of servos, DC motors, LED lights, or even a combination of both.



Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface
You want to make a cool robot, maybe a hexapod walker, or maybe just a piece of art with a lot of moving parts. Or maybe you want to drive a lot of LEDs with precise PWM output. Then...
https://www.adafruit.com/product/815

We are powering the servo from an external 5V 2A switching power supply connected to the terminal block on the breakout board via a DC adapter.

Why wouldn't we use the +5V supply on the Raspberry Pi instead?

Switching directions on the servo can cause a lot of noise on the supply, and the servo(s) will cause the voltage to fluctuate significantly, which is a bad situation for

the Pi. It's highly recommended to use an external 5V supply with servo motors to avoid problems caused by voltage drops on the Pi's 5V line.

---

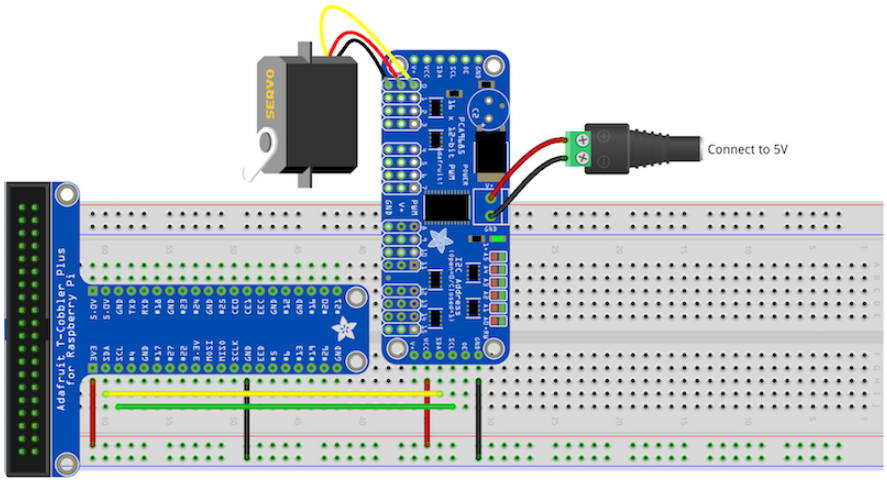
1 x Power Supply https://www.adafruit.com/product/276

5V 2A (2000mA) switching power supply.

---

1 x Female DC Power Adapter https://www.adafruit.com/product/368

2.1mm jack to screw terminal block.



The VCC input is the power supply for the IC (3.3.V). The V+ input is the supply for the servo motors (typically 5V). Be sure not to confuse the two or you'll end up with a burnt Pi.

Make the following connections between the Raspberry Pi and the PCA9685:

- **Pi 3.3V** to **Power Rail**
- **Pi GND** to **Ground Rail**
- **3.3V** to **PCA9685 VCC**
- **GND** to **PCA9685 GND**
- **Pi SDA** to **PCA9685 SDA**
- **Pi SCL** to **PCA9685 SCL**

Make the following connections between the servo and the PCA9685's Channel 0:

- **Servo Black/Brown** to **PCA9685 GND**
- **Servo Red** to **PCA9685 V+**
- **Servo Yellow** to **PCA9685 PWM**

Finally, connect the PCA9685's screw terminal to the external power supply:

- **PCA9685 V+** to the **Power Supply + (positive)**
- **PCA9685 GND** to the **Power Supply - (negative)**

Python Setup

If you're following along with a Raspberry Pi, Beaglebone or any other supported small linux computer, we'll use a special library called **adafruit_blinka** (https://adafru.it/BJS) (named after Blinka, the CircuitPython mascot (https://adafru.it/BJT)) to provide the layer that translates the CircuitPython hardware API to whatever library the Linux board provides. It's CircuitPython, on Pi!

If you haven't set up Blinka and the Adafruit IO Python Library yet on your Raspberry Pi, follow our guide:

- Blinka + Adafruit IO Setup (https://adafru.it/BMB)

The latest Raspbian (currently this is `Stretch`) is required for the installation of Adafruit IO + Blinka.

Enable I2C

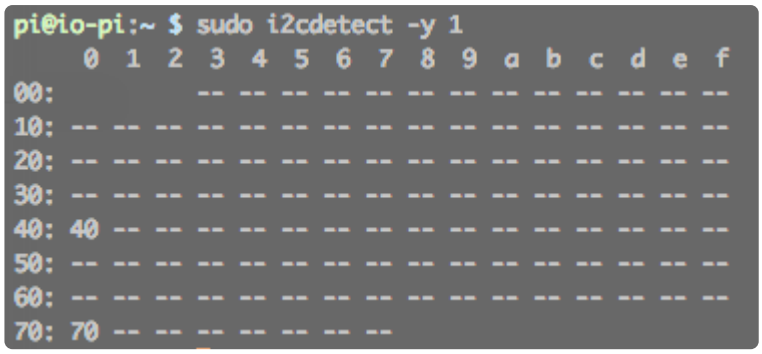
We use two pins on the Pi (SDA/SCL) to communicate over I2c with the PCA9685. You only have to do this step once per Raspberry Pi, the I2C interface is disabled by default.

- Enabling I2C (https://adafru.it/dEO)

Once you're done with this and have rebooted, verify you have the SPI devices with the command:

```
sudo i2cdetect -y 1
```

If your PCA9685 Breakout is wired up correctly, it'll show up at 0x40:



Install the CircuitPython-PCA9685 Library

You'll also need to install a library to communicate with the PWM breakout. Since we're using Adafruit Blinka (CircuitPython), we can install CircuitPython libraries on our Raspberry Pi. In this case, we're going to install the CircuitPython-PCA9685 library.

Run the following command to **install the CircuitPython-PCA9685 library**:

```
pip3 install adafruit-circuitpython-PCA9685
```

Note: While this package is dependent on two other packages **adafruit-circuitpython-busdevice** and **adafruit-circuitpython-register**, they're installed with the package.

Install the CircuitPython-Motor Library

Controlling the PCA9865's channels directly is not an easy task. You'll need to manually set the duty cycle of each channel. Luckily, "there's a CircuitPython library for that".

CircuitPython-Motor is a helper library for easily controlling motors, servos, and PWM-based outputs.

Run the following command to install the **CircuitPython-Motor** library:

```
pip3 install adafruit-circuitpython-motor
```

Python Code

The imports for this guide are similar to the other guides in this series involving analog PWM output (Color (https://adafru.it/uFC), Analog Output (https://adafru.it/uFE)), except this time it uses an extra library: **adafruit-circuitpython-motor**

```
import time # system
from board import SCL, SDA # blinka
from busio import I2C
from adafruit_pca9685 import PCA9685 # PCA Module
from Adafruit_IO import Client, Feed, RequestError # adafruit io
from adafruit_motor import servo # servo library
```

By default, the servo channel is on channel 0 of the PCA9685. If you'd like to change this, you can do so by modifying the SERVO_CHANNEL variable at the top of the code:

`SERVO_CHANNEL = 0`

Before we run the script, we'll need to change `ADAFRUIT_IO_USERNAME` and `ADAFRUIT_IO_KEY` to the username and key for your Adafruit IO account.

```
# Set to your Adafruit IO username.
ADAFRUIT_IO_USERNAME = 'YOUR_IO_USERNAME'

# Set to your Adafruit IO key.
ADAFRUIT_IO_KEY = 'YOUR_IO_KEY'
```

Next, we set up the PCA9685 by creating an I2C bus, and passing that into a PCA9685 class. Then, we set the PWM frequency.

```
i2c_bus = I2C(SCL, SDA)
pca = PCA9685(i2c_bus)
pca.frequency = 50
```

Then, we instantiate a servo object called **my_servo:**

`my_servo = servo.Servo(pca.channels[SERVO_CHANNEL])`

in the **while True** loop, we first grab the servo feed value. Then, we compare it against the previous feed value. If it the feed value is different from the previous value, we set the servo angle to the feed's value.

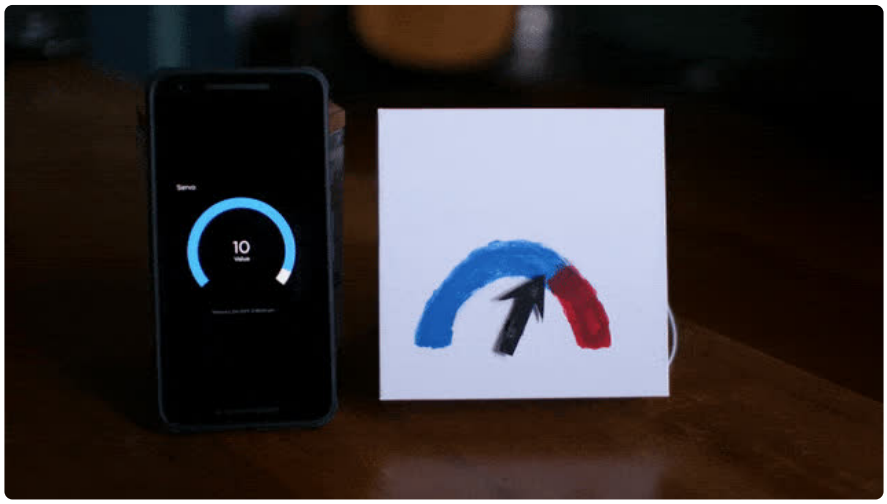
```
while True:
    # grab the `servo` feed value
    servo_angle = aio.receive(servo_feed.key)
    if servo_angle.value != prev_angle:
        print('received &lt;- ', servo_angle.value)
        # write the servo to the feed-specified angle
        my_servo.angle = int(servo_angle.value)
    prev_angle = servo_angle.value
    # timeout so we don't flood IO with requests
    time.sleep(0.5)
```

Unlike the Arduino example code, we don't need to handle the servo going past it's maximum or minimum angle, the **adafruit-circuitpython-motor** library handles that for us.

Change the slider value on your Adafruit IO dashboard, and you should see the values output from your terminal:

```
received &lt;-  120 Degrees
received &lt;-  70 Degrees
received &lt;-  30 Degrees
received &lt;-  90 Degrees
```

You should also observe the servo change position depending on what value you send:



Adafruit IO FAQ

Encountering an issue with your Adafruit IO Arduino Project?

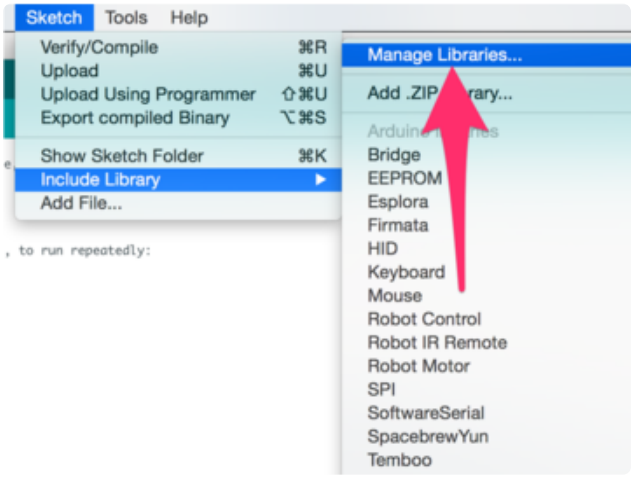
If you're having an issue compiling, connecting, or troubleshooting your project, check this page first.

Don't see your issue? Post up on the Adafruit IO Forum with your issue (https://adafru.it/plC).

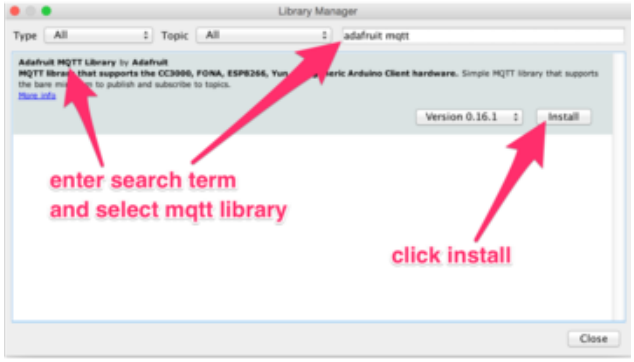
I encounter the following error when compiling my sketch:

`fatal error: Adafruit_MQTT.h: No such file or directory, #include "Adafruit_MQTT.h"`

The Adafruit IO Arduino library is dependent on our Adafruit IO MQTT Library.



To resolve this error, from the Arduino IDE, navigate to the **Manage Libraries...** option in the **Sketch -> Include Library** menu.



To resolve this error, from the Arduino IDE, navigate to the **Manage Libraries...** option in the **Sketch -> Include Library** menu.

My Serial Monitor prints "..." endlessly after the "Connecting to Adafruit IO" message

Your board is not connecting to Adafruit IO, but why? Let's find out:

First, check in `config.h` that you have the correct `IO_USERNAME`, `IO_KEY`, `WIFI_SSID`, and `WIFI_PASS` are set correctly.

Next, we're going to modify the while loop which waits for an IO connection in your sketch. Change the line in the status check loop from `Serial.println(.);` to `Serial.println(io.statusText());`

```
// wait for a connection
while(io.status() < AIO_CONNECTED) {
Serial.println(io.statusText());
delay(500);
}
```

Verify and re-upload the sketch. If you're receiving a **Network disconnected** error message, the board is not able to talk to the internet. Re-check your hardware, connections, and router settings.

If it's still not showing **Adafruit IO connected**, check the IO status on the Adafruit Status page (https://adafru.it/Oc0) to make sure the service is online.

My data isn't displaying, is Adafruit IO's {service/MQTT/API} down?

Possibly - you can check IO status on the Adafruit Status page (https://adafru.it/Oc0).

Is my data being sent properly? Am I sending too much data?

There's a monitor page built-into Adafruit IO (https://adafru.it/DOK) which provides a live view of incoming data and error messages. Keep this page open while you send data to your Adafruit IO devices to monitor data and errors.